

Escuela de ingeniería en computación
Ingeniería en computación
IC1803 - Taller de programación

Numeradores y denominadores para un cociente dado

Proyecto 1
Profesor
Alex Brenes

Aaron Pérez Alfaro
a.perez.1@estudiantec.cr
2024185088

Alajuela, Costa Rica
Abril 2024

Resumen ejecutivo

El proyecto consistía en la solución a un problema planteado, el cual se necesitaba encontrar dos números donde el cociente dé como resultado un número introducido por el usuario (C), y a su vez, los resultados tienen que tener la condición de que ambos números pueden tener un mínimo de cuatro dígitos y un máximo de cinco, además, los dos números entre ellos no pueden sobrepasar la cantidad de dígitos repetidos especificados por el usuario, donde R da la cantidad de recurrencias permitidas.

Esta sería la forma representada, donde del dígito 0 al 4 es un número y del 5 al 9 es otro número.

$$\frac{d_0d_1d_2d_3d_4}{d_5d_6d_7d_8d_9} = C$$

El usuario a la hora de utilizar el programa, debe introducir cuantos casos de prueba va a necesitar, donde T es la cantidad de casos y tiene que ser mayor a 0 y menor o igual a 100. Seguido tendrá que colocar de forma consecutiva el número que desea encontrar y la cantidad de recurrencias de los dígitos y así la cantidad de casos de prueba que introdujo. Y el resultado se entrega por consola, donde los resultados aparecen de menor a mayor con respecto al numerador, donde cada caso está separado por un salto de línea y si el caso no tiene resultado simplemente deja una línea en blanco.

El algoritmo principal de este problema sería en función del concepto de divisibilidad, ya que, en el programa se deben buscar dos números que divididos den C . El concepto de divisibilidad utilizado en el código es:

$$a|b \Rightarrow b = ac$$

Donde c es el número que introdujo el usuario para encontrar y a es el número a buscar por iteración, gracias a que al ir probando número por número en a y multiplicándolo por c vamos a ir encontrando todos los posibles numeradores, entonces la división daría como resultado b , osea, encontramos los números que divididos dan como resultado C . Seguido, verificamos que los dos números cumplan con el limitante de R , osea cuantas veces se repiten los dígitos de ambos números en conjunto.

En resumen, la solución fue de ir iterando hasta que la multiplicación de c con el valor de a sobrepase 99999; dentro del loop hacemos la multiplicación de a con c y lo guardamos en la variable b . Luego se comprueba que ambos números cumplen con los requisitos (a, b) y si pasan, se inserta una cadena con el formato *numerador/denominador* = c y en este código se representaría como $b/a = c$, con un salto de línea en la cadena, así hasta terminar con

el ciclo. Y por último se entrega la cadena de texto con todos los resultados para un caso en específico.

La sección de comprobar los números, sería básicamente, si los números son mayores o iguales que 1000 y menor a 10000, osea que solamente tienen cuatro dígitos, entonces se le agregan un cero a cada número. Después se unen los dos números en una lista y se comprueban todos los elementos de la lista, para saber si la cantidad de dígitos no es mayor que R y sus valores de retorno son: true o false, false si sobrepasa R y true si se cumple con la condición. Y por último los resultados de todos los casos de prueba, serán imprimidos por consola.

Con respecto a los resultados, se utilizó en la gráfica (figura 1) con $C = 5$ y $R = 5$, donde se probó de 0 a 200 casos de prueba, mostrando la eficiencia del código para obtener los resultados, donde para el caso de prueba 200 dió como resultado 9 segundos, con el límite de tiempo de 15 segundos. Asimismo, en la carpeta casos de prueba, se encuentran un total de 10 casos donde se prueba el código con diferentes datos de entrada, siendo graficada con la biblioteca *matplotlib* [1] para agilizar la forma de graficar los resultados obtenidos según los casos solicitados.

En relación a la eficiencia del programa, en el código en forma de comentarios vienen explicados de forma detallada la máxima cantidad de iteraciones que realiza cada función y su respectivo *Big O*, siendo T el que marque la complejidad de tiempo del programa, ya que, a medida que los casos de prueba aumente, también aumenta el tiempo de ejecución. Y por ejemplo: la función *foo* en el while, sin contar con la verificación del *todo* (comprobar los dígitos) hace en el peor de los casos 98999 iteraciones, porque iniciamos en 1000 y vamos contando de uno en uno porque $a * 1$ siendo a el número iterado, hasta que llegamos hasta el número 99999, osea se hicieron 98999 iteraciones. Ya para la función *todo* itera unas 100 veces, dado que, el count tiene como *Big O* la cantidad de la longitud de la lista, osea 10, y a su vez hay un forma que itera la lista. Por lo cual hace $10 * 10 = 100$ iteraciones, cada vez que se llame a la función.

A lo largo del proyecto surgieron varios inconvenientes, por ejemplo: no se encontraba una forma eficiente de buscar los dos números, ya que, de un principio se trataba de buscar ambos números a la vez, otro problema fue el encontrar una forma de entregar los resultados al usuario, ya que, al inicio se imprimía por consola y eso generaba que el programa aumentara mucho el tiempo de ejecución, por lo cual, se optó por cambiar la salida estandar por insertar en un archivo.

Introducción

El proyecto consistía en la creación de un programa bajo los principios de *backtracking* [2], el cual encontraba pares de números donde su cociente daba como resultado un número natural específico. Con la limitante de tiempo de 15 segundos de ejecución del programa.

A lo largo de la documentación se explica las funciones y algoritmos utilizados en el programa, además se evidencia el proposito de cada función implementada, y con varios archivos de casos de prueba con una gráfica mostrando los tiempos de ejecución del programa, donde muestra la eficiencia del código. También se comentan las complicaciones a lo largo del desarrollo del proyecto y su solución a dicho problema. Y por último se muestran los aprendizajes que se obtuvieron mientras se desarrollaba el proyecto.

Marco teórico

Para la resolución del problema, utilicé el ide visual studio code, con el lenguaje python3, ya que, visual es muy amigable con su entorno, además, de las facilidades, por la implementación de herramientas como el apartado de extensiones para descargar utilidades de la comunidad y también su sección de control de código, para subir los archivos a github. Y se utilizó python por su entorno fácil de utilizar haciendolo muy sencillo de aprender y empezar con la programación y su buen manejo con números inmensos y su gran productividad, ya que, hace al desarrollador escribir programas con menos líneas de código.

Las bibliotecas usadas fueron las de *time* y *Matplotlib*, la primera está implementada en el paquete de python, y su función (en este código) fue de medir el tiempo de ejecución, ya que, era tedioso el estar utilizando la funcionalidad de *linux* o la web *ideone*. Y la segunda, fue utilizada para crear la gráfica para introducirla en la sección de resultados de las pruebas. Estas bibliotecas en la versión de entrega, serán eliminadas, osea solo fueron implementadas en el apartado de desarrollo, para mayor comodidad al desarrollador a la hora de escribir el código.

Toda la documentación, programa, casos de prueba, serán subidas a la página web *github*, debido a su amigable interfaz y el compartir código con terceros. El repositorio de *git* donde se almacenarán toda la documentación será [Proyecto-1-taller-programacion](#)

-A Historia de python:

Python fue creado por Guido van Rossum, lanzado en octubre del 2000 con python2 y tiempo después

lanzaron python3. Al principio inició como un proyecto de afición para mantenerse ocupado durante las vacaciones de Navidad. [3]

Algunas características del lenguaje son:

- Lenguaje interpretado
- Lenguaje fácil de utilizar
- Lenguaje tipeado dinámicamente
- Lenguaje de alto nivel
- Lenguaje orientado a los objetos

-B Historia de visual studio code:

Visual studio code es un editor de código fuente desarrollado por Microsoft. Anunciado el 29 de abril de 2015 por Microsoft. Este editor cuenta con funcionalidades sorprendentes de gran provecho para cualquier desarrollador, para cualquier lenguaje de programación imaginado, con una gran comunidad. [4]

Algunas características del editor son:

- Multiplataforma
- Uso del control de versiones
- Extensiones

-C Historia de github:

Esta plataforma fue creada por Chris Wanstrath , PJ Hyett , Tom Preston-Werner y Scott Chacon en el año 2008, y cumple un rol muy destacado en los grandes proyectos que se han y se vienen desarrollando en la actualidad. Ya que ha permitido en su entorno en facilitar el trabajo grupal, haciendo que en ella se permitan llevar un control de las versiones del proyecto, asimismo sirve como almacenamiento de código y archivos [5].

-D Historia de git:

Lanzado en 2005 por Linus Torvalds, puesto que, en su tiempo existía un programa llamado BitKeeper dejó de ser gratuita y la comunidad empezó a desarrollar su propio programa. Ellos se plantearon los objetivos de:

- Velocidad
- Diseño sencillo
- Gran soporte para desarrollo no lineal
- Completamente distribuido
- Capaz de manejar grandes proyectos

Y a día de hoy es de las aplicaciones más utilizadas por los desarrolladores de todo el mundo, donde se les permite el controlar las versiones de sus proyecto, trabajo en equipo, la posibilidad de volver a la versión anterior por si hubo algún error [6].

Descripción de la solución

Detalle del algoritmo

La función principal es, por medio de un `while` ir sumándole 1 a a , e ir probando la multiplicación de a con c hasta que ese producto sobrepase 999999 que sería el límite, ya que, después de eso, los números serían de más de cinco dígitos. Cada división encontrada se introducía a una lista, por lo cual, cuando tuviera todos los resultados insertados retornaba un string con todos los elementos encontrados

El código de *E/S* se encarga de obtener por medio de inputs los valores que se introducen por consola. Para el primer input se recibe T , el cual son el número de casos a introducir por medio de un `for`, el cual permite introducir los valores de C y R juntos, por lo cual, se tiene que usar la función `split()` para separar los elementos insertados y almacenarlos en las variables y esos mismos introducirlos a una lista, donde se tienen todos los valores de C y R introducidos. Luego por medio de un `for` se iteran los casos y se va llamando la función con los valores, y a su vez se van agregando los resultados a una lista, para luego esa lista iterarla, e ir imprimiéndola por consola.

El programa consta de dos funciones: *todo* y *foo*.

- **foo:** `foo` es la función principal, en el cual solo se reciben dos parámetros: C y R , el primero es el número a encontrar por medio del cociente de dos números y el segundo es el total de dígitos que se pueden repetir.

Esta función se encarga de por medio de un `while` ir buscando el numerador, con el cual multiplicando a c va encontrando los números necesarios, por medio del concepto de divisibilidad. E introduce en una cadena, los resultados a medida que los encuentra por medio de la función *todo*. Y por último, retorna la cadena de los elementos obtenidos, con el formato de cada elemento $\text{numerador}/\text{denominador} = C$. Un ejemplo de los valores que retorna *foo* sería: $99776/49888=299774/49887=2$. Ya que de por medio de los dos resultados, existe un salto de línea.

- **todo:** se encarga de la verificación de los dígitos de los números, por ejemplo, dependiendo de R , el código permitirá una repetición máxima de dígitos, por numerador y denominador. En resumen de cómo funciona la función es: primero, si el numerador o el denominador tienen 4 dígitos, por medio de una multiplicación de 10 se le suma un 0 a los dígitos, ejemplo: 1234 pasa a 12340, ya que, aun que no se vea que el

número tenga un 0, en profundidad si lo tiene. Segundo, se unen los dos números en una lista, e iteramos todos los elementos, y haciendo un `count(i)` para buscar cuantas veces ese dígito se repetía entre el numerador y el denominador.

Complicaciones

Al principio no encontraba una forma eficiente de buscar los números que divididos den C , hasta que encontré el concepto de división, tal que, $a|b \Rightarrow b = aq$ siendo q el C , y por despeje, solo tendríamos que buscar un número que multiplicado con c dé b .

Otra complicación encontrada sería la forma de verificar los dígitos de los números del numerador y denominador, ya que, los algoritmos que utilizaba eran ineficientes, donde se creaban variables innecesarias, loops que podían simplificarse en solo uno, entre otros.

Y además, encontré la ineficiencia de imprimir por consola usando el `print()`, ya que, para casos pequeños es óptimo, pero cuando tiene que escribir por consola grandes cantidades de outputs, hace que el tiempo de ejecución se incremente. Pero es curioso que si imprimimos solo la lista de los resultados, así: `[1,2,3,4]` es muy rápido, pero cuando tiene que imprimir cada elemento de esa lista, osea: 1, 2, 3, 4; todos en líneas diferentes, se tarda mucho en mostrarlos. Ya que, el código tiene que solicitar permisos de escritura por consola, y eso causa que el tiempo se dispare a mayor cantidad de prints solicitados [7].

Resultados de la pruebas

En la carpeta casos de prueba se encontrarán todos los casos de prueba realizados, con su respectivo tiempo de ejecución, sus inputs y resultados. Así mismo se mostrará una gráfica con varias casos prueba.

En esta gráfica (figura 1) se muestra el tiempo de ejecución para los casos de prueba de 1 a 200 para $C = 5$ y $R = 5$. Cabe recalcar que el tiempo de ejecución introducido en la gráfica es aproximado y cada computadora puede presentar resultados de tiempo distintos aproximados.

Esta gráfica fue creada usando la biblioteca *matplotlib* con los valores introducidos manualmente [1].

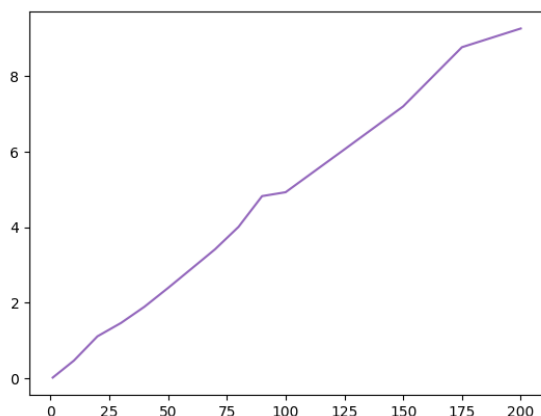


Figura 1. x: segundos, y: casos de prueba

Conclusiones

En relación a los resultados obtenidos, introducidos en los casos de prueba, se puede concluir que el tiempo de ejecución para uno o varios C y R , crece a medida que aumenta T , siempre en el peor de los casos, ya que, puede que para un C y R no tengan resultados.

Con respecto al programa, se pudo comprobar la eficacia de los inputs introducidos, arreglando el problema de los prints, intercambiándolo por introducir el output en un archivo de texto, evitando que el tiempo de ejecución aumente por el excesivo uso de los prints.

Otro punto importante, para $C = 1$ y $R = 5$, sería el peor de los casos en los que se pondría a prueba el programa, ya que, alrededor haría como 9899900 bucles, por el primer while donde hace 98999 iteraciones en el peor de los casos y la función *todo* siempre haría 100 iteraciones, osea $98999 * 100$. Eso sería solo donde el T es 1, y aumenta a medida que aumenta el T tipo $O(t)$.

Aprendizajes

1. Aprendí un mejor manejo de las listas y métodos en python [8].
2. Aprendí a priorizar la eficiencia y rendimiento del programa.
3. Comprendí lo importante de la utilización de algoritmos optimos para la creación de la más mínima función, ya que, podría afectar la eficiencia en tiempo de los programas.
4. Aprendí a implementar una función para el control de tiempo de ejecución de los programas [9].
5. Aprendí a formatear los strings en python [10].
6. Aprendí lo ineficiente de usar los prints con grandes listas de elementos.

Referencias

- [1] P. Jorge. "Graficar en Python con Matplotlib y NumPy. Numyhton." (2016), dirección: <https://numyhton.github.io/posts/2016/02/graficar-en-python-con-matplotlib-y/>.
- [2] S. D. M. Aibin. "Backtracking Algorithms." (2024), dirección: <https://www.baeldung.com/cs/backtracking-algorithms#:~:text=Backtracking%20is%20an%20algorithmic%20technique,satisfy%20them%20will%20be%20removed.>
- [3] C. P. Santiago De Cuba D. "El lenguaje de programación Python." (n.d.), dirección: <https://www.redalyc.org/pdf/1815/181531232001.pdf>.
- [4] F. Flores. "Qué es Visual Studio Code y qué ventajas ofrece." (2022), dirección: <https://openwebinars.net/blog/que-es-visual-studio-code-y-que-ventajas-ofrece/>.
- [5] N. Colectiva. "Que es GitHub, Historia y otros Detalles," Blog de Programación y Desarrollo." (2019), dirección: <https://blog.nubecollectiva.com/que-es-github-historia-y-otros-detalles/>.
- [6] Git. "Una breve historia de Git." (2024), dirección: <https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Una-breve-historia-de-Git>.
- [7] J. Half. "How to create a new text file using Python. Stack Overflow." (2024), dirección: <https://stackoverflow.com/questions/48959098/how-to-create-a-new-text-file-using-python>.
- [8] W. schools. "Python List/Array Methods." (n.d.), dirección: https://www.w3schools.com/python/python_ref_list.asp.
- [9] E. libro de python. "Medir tiempo de ejecución." (n.d.), dirección: <https://ellibrodepython.com/tiempo-ejecucion-python>.
- [10] W. schools. "Python String format() Method." (n.d.), dirección: https://www.w3schools.com/python/ref_string_format.asp.